A NEW AGILE PROCESS FOR WEB DEVELOPMENT

Vinícius Pereira and Antonio Francisco do Prado
Department of Computing, Federal University of Sao Carlos, Sao Carlos – SP, Brazil

Keywords: Agile Process, User Story, Navigation Model, Web Development.

Abstract: In this paper is described an agile methodology for Web development based on User Stories. The main objective in this methodology is to have a more real relationship among application code and requirements. Thus, the development team and the user may come to have a greater understanding during the application development process. It is divided in three disciplines, each one refining the User Stories, from requirements specification using the Navigation Model and Story Cards until the execution of these User Stories to guide the coding. The team can also use these Stories as acceptance tests, which represent the user behaviour when using the system. With all this, in the end the development team may have more guarantees that the Web application represents what the user wants.

1 INTRODUCTION

In the early versions of the Web there was little room for its users to publish information and interaction was very restrictive. At that time, the Web consisted specially of static HTML pages and/or some very few Java applets. With the Web 2.0 (O'Reilly, 2006) new technologies began to gain the moment, opening the Web for social collaboration. Good examples of this are the social networks (Recuero, 2004), with the growing focus on collaborative software – groupware – (Carstensen, Schmidt, 1999). Collaborative software is designed to facilitate interactions between groups of individuals that share a common objective. This is just to show how the Web (with the Internet) changes the way that people interact with the computer and others devices with Web interfaces.

Considering the growth tendency of collaborative software in many areas, like education, trading, healthcare and others, a more participative and agile approach becomes necessary for corporations and institutions. These ideas, together with the new techno-logies available today are promising to accelerate the development process. A good example to illustrate that tendency is Twitter, a social network focused on the concept of micro blogging that allows its users to pu-blish personal updates and see updates from the others in a computer or in a smartphone, for example.

This article presents a new agile methodology for the development of Web applications using the concept of User Stories and making sketches of the Navigation Model to guide the construction and details these Stories in conversations with the user. Modern tech-niques and concepts are the basis for the development of this agile methodology.

2 CONCEPTS AND TECHNIQUES

This section presents the main concepts and techniques behind the proposed methodology for developing Web application. The core concept is the User Story (Cohn, 2004). There are also brief descriptions regarding Test Driven Development (Beck, 2003), Behavior Driven Development (North, 2006), Web Engineering (Press-man, Lowe, 2009), Scrum (Schwaber, 2004) and eXtreme Programming (Beck, Andres, 2004). Web Engineering is the starting point that led the studies to other techniques and concepts. BDD is the inspiration model for the methodology. The use of TDD is heavily encouraged in one of the disciplines of the agile metho-dology that is proposed here. Scrum – principally – and XP ideas are reused in some points in this process.

2.1 User Story

User Stories describes functionally what it is requested and valuable for the User. In a User Story there are 3 C's which are: Card, Conversation and Confirmation; and follows the principle of INVEST:

Pereira V. and Francisco do Prado A..
A NEW AGILE PROCESS FOR WEB DEVELOPMENT.
DOI: 10.5220/0003504501770187
In *Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2011), pages 177-187
ISBN: 978-989-8425-57-7

Independent, Negotiable, Valuable for the user, Estimable, Small and Testable. One of the C's cited previously, the Story Card is the User Story written and formalized. From this card can be seen the other two C's.

The idea of the User Story be written in a card instead of another media have the purpose to maintain the principle of Small (so the story gets short). If a User Story exceeds the card limit maybe it is time to break it. The important is that there is no limit to write User Stories since they are in the pattern (Cohn, 2004).

An informal example is: "Students can purchase parking passes". In his book Mike Cohn suggests a more formal approach to writing User Stories (Cohn, 2004). He suggests the format: As a *"role"* I want *"something"* so that *"benefit"*. This approach helps to think about why a certain feature is built and for whom, and as a result is the approach that is typically taken. The same example in this approach is: "As a student I want to purchase a parking pass so that I can drive to school". As can be seen, the formality brings a greater understanding of the problem.

Agile methodologies favor face-to-face communication over comprehensive documentation and quick adaptation to change instead of fixation on the problem. User Stories achieve this by: (1) they represent small chunks of business value that can be implemented in a period of days to weeks; (2) needing very little maintenance; (3) allowing developer and the client representative to discuss requirements throughout the project lifetime; (4) allowing projects to be broken into small increments; (5) being suited to projects where the requirements are volatile or poorly understood; (6) require close customer contact throughout the project so that the most valued parts of the software gets implemented.

Some of the limitations of User Stories in agile methodologies are: (1) they can be difficult to scale to large projects; (2) they are regarded – usually – as conversation starters and nothing more.

2.2 Test Driven Development

Created by Kent Beck (Beck, 2003), TDD is an approach to deal with analysis and specifying behavior based on automated tests. Test Driven Development introduces the concept of Red-Green-Refactor: (1) write a test and watch it fail; (2) write the minimum code necessary to make the test passes; and (3) apply refactoring with design patterns (Fowler, 1999) to eliminate redundancies or duplicated codes.

Kent Beck considers that TDD encourages simple code design and increases confidence in the final product (Beck, 2003). With TDD, according Feathers, programmers can improve legacy code without the fear of changing the existing behavior (Feathres, 2004).

In this approach, a test is a piece of software that has two main goals: (1) specification: establishing a rule that the software has to follow and (2) validation: verify that the rule is properly implemented by the software. With this, it is possible to generate clean code, which is the code that reflects exactly what it had been designed to do, without trickery or obscurity (Martin, 2008).

The main advantages of using TDD are: (1) the code has less coupling and greater cohesion; (2) the code has greater quality because it is fully tested; (3) refactoring can be executed without fear of breaking behavior; (4) it is possible to know clearly when a task is done – when the corresponding test is passing; (5) the test suite serves as a basis for automated regression tests without need for further development; (6) the vast majority of the bugs are found earlier, which make the effort to fix them cheaper.

2.3 Behavior Driven Development

BDD, created by Dan North, is an agile technique that encourages the collaboration between developers, quality assurance people and business people during the process of software development. Briefly, in BDD the client/user defines how the application should behave by writing an automated test to verify it. After that, the code necessary for that behavior is implemented. This sort of test is considered an acceptance and/or functional test.

The main difference between BDD and TDD is that on BDD the focus is on the business rules to be fulfilled by the software. It might sound like a purely conceptual difference but in reality it isn't. For example: "the initial screen should list all customers" isn't the same as "the method *HomeController.index* needs to create a variable called customers". The first approach is understandable by a user, while the second one is geared towards a programmer's audience. The first is an example of the definition's behavior of a screen, something that a user could say naturally. The second is an example of how a programmer might read the source code necessary to make that functionality works.

BDD offers some advantages: (1) increases the integration between final users, testers and developers, since all of them speak the same language; (2) even when testers and developers are in

completely different teams, they can work together to describe "what" is to be solved by the software. Writing User Stories is a good way to define "what" should be done, because it can be realized with the help of the final user or at the very least validated by him, who can understand what will be done according the content of these Stories; (3) User Stories provide test cases, automated tests, and project specification; and (4) the User Stories become executable. With this, it can be said that the final user is able to "write the code" for the acceptance tests. Finally, besides encouraging better quality code, BDD decreases the overall amount of work for the team and improves communication, which is essential for a more agile development process.

2.4 Web Engineering

The World Wide Web has become a crucial platform for the delivery of a variety complex and diversified corporative applications in many business domains. Besides its distributed aspect, these Web applications require constant improvements in usability, perfor-mance, security and scalability. However, the vast majority of the aforementioned applications are still being developed with an ad hoc approach, contributing for usability problems, maintenance, quality and reliability (Pressman, 2001). Even considering that the Web development can benefit from methodologies inherited from other areas, it has very specific charac-teristics that require a special approach. Among these characteristics there are the network traffic, parallelism, unpredictable load, performance, availability, focus on data, context sensitivity, continuous evolution, urgen-cy, security, and aesthetics (Pressman, Lowe, 2009).

Web Engineering allows a systematic, disciplined and quantifiable approach for high quality development focused on Web applications (Ginige, Murugesan, 2001). Focusing on methodologies, processes, techni-ques and tools applied in different abstraction levels, from the conception to development, evaluation and maintainability.

The principles of Web Engineering include: (1) a different and unique process of Web development (Kappel, et al., 2003); (2) multidiscipline. It is very unlikely that a single discipline can offer a complete theoretical basis, with knowledge and practice to guide the Web development (Deshpande, Hansen, 2001); (3) the continuous evolution and the management of the life's cycle of a software (cycles as short as possible) when compared to the traditional development methods; and (4) the applications can be pervasive and non-trivial. The Web perspective as a platform will continue to grow and should be addressed as one.

2.5 Scrum

Scrum is a framework for agile software development that is iterative and incremental. Initially it was con-ceived as a project management style for the automo-bilist and consumables industries. They noticed that on projects using small multidisciplinary (cross-functio-nal) teams where the results were considerably better. In 1995, Ken Schwaber formalized the definition of Scrum (Schwaber, 2004) and helped to introduce it to the software development worldwide.

The primary function of Scrum is to be used for management of software development projects, it can be used to run software maintenance teams, or as a general project/program management approach. It can be too, in theory, applied to any context in which a group of people needs to work together to achieve a common goal.

Scrum has three main roles: (1) the *Scrum Master*, who maintains the processes; (2) the *Product Owner*, who represents the stakeholders and the business; (3) the *Team*, a cross-functional group of about 7 people who do the actual analysis, design, implementation, testing, among another tasks.

During each *Sprint*, typically a two to four week period, the Team creates a potentially shippable product increment. The set of features that go into a sprint come from the *Product Backlog*, which is a prioritized set of high level requirements of work to be done. Which backlog items go into the Sprint is determined during the *Sprint Planning Meeting*. During this meeting, the Product Owner informs the Team of the items in the Product Backlog that he or she wants completed. The Team then determines how much of this they can commit to complete during the next Sprint.

During a Sprint, no one is allowed to change the *Sprint Backlog*, which means that the requirements are frozen for that sprint. Development is time boxed such that the Sprint must end on time; if requirements are not completed for any reason they are left out and returned to the Product Backlog. After a Sprint is completed, the Team demonstrates how to use the software to the User.

2.6 eXtreme Programming

Another agile methodology, XP is a method for small and medium teams that will develop software with vague requirements and in constantly changes, which

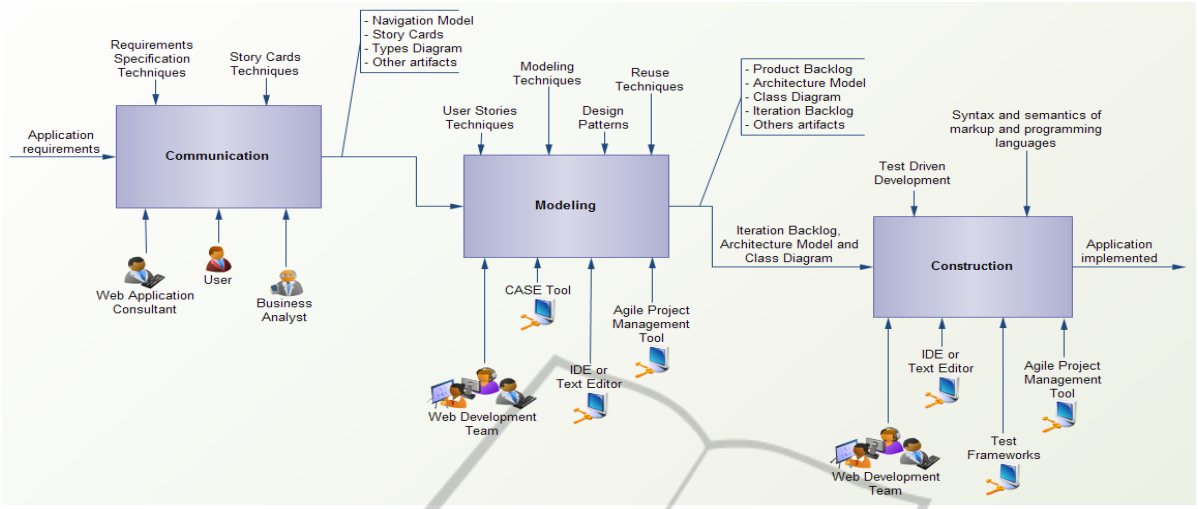

Figure 1: The process and the artifacts.

is intended to improve software quality and responsiveness to changing customer requirements. XP adopts the strategy of constant monitoring and execution of several minor adjustments during the development of software. The four core values of XP methodology are communication, simplicity, feedback and courage.

From these values, has as its basic principles: rapid feedback, assume simplicity, incremental change, em-bracing change and quality work.

Among the control variables in projects (cost, time, quality and scope), there is an explicit focus on the scope. XP recommends the prioritization of features that represent the possible highest value for business. Thus, the more necessary for the reduction of scope is, the less valuable features will be postponed or canceled.

The XP encourages the control of quality as a project variable, because the small short-term gain in productivity, while decreasing quality, is not compensated for losses (or hindrance) in the medium and long term.

3 A NEW AGILE PROCESS FOR WEB DEVELOPMENT

The process proposed here includes three disciplines: *Communication*, *Modeling* and *Construction*. The process follows the concept of using User Stories to get the requirements and use these Stories in all the development process, like in Behavior Driven Development, but showing another form to obtain the User Stories: making use of the Story Cards and the Navigation Model to complement them; and how to apply all this in the process. Moreover, the process makes uses of ideas and concepts of the Web Engineering proposed by Pressman as well as some ideas of Scrum and XP. The Figure 1 shows in a diagram SADT – Structured Analysis and Design Technique – (Marca, McGowan, 1988), the disciplines of the process which are executed in each cycle of the process.

It is suggested to the consultant that he gets as much information as possible on the requested Web application in early conversations with the user. This amount of information will be vital for the development team to analyze and project the time and effort to accomplish the implementation of the system. All the people involved in the project should realize that having “as much information as possible” does not imply implement everything at once. The requirements specified in this information will be broken into shorter (and functional) cycles of iterations. More details about this will be discussed throughout this paper.

Another fact that can happen is that some of the features will be only requested when the user interact with the application. The team should have in mind that a Web application will be always evolving. Test the ideas and collect the feedback for these tests before going ahead. The earlier the problems and/or mistakes were discovered, the fewer resources and time will be spent to fix them.

In general, this type of development involves a large quantity of changes along the project. The process is based in iterations according to the User Stories originated in the conversations with the user. The prototypes developed in each iteration constitute mini-projects that slowly include all the functionality

of the software, based in the functionalities and characteristics that these User Stories requests.

For better understanding, a clarification is needed. The term "Story Card" is used in Communications to express a User Story in card format, like Mike Cohn proposes in his book. And the term "User Story" in Modeling and Construction disciplines, is used to express the format proposed by Dan North, with some key words that will be useful in these disciplines.

To assist in the presentation of the disciplines is used as an example of a Web application a kind of social software, the groupware (specifically a wiki). Therefore, all figures show a part of the specification of a wiki to help illustrate the description of artifacts.

3.1 Communication

In this discipline, the main focus is to extract information in the conversations with the user. The concern is to get as much information as possible about how the system should behave. To assist in require-ments specification, the proposed approach makes use of Story Cards and the Navigation Model.

There is no rule saying which of the two types of artifacts should be generated first, because the process assumes that they are complementary. At the end of the conversation with the User, the Story Cards and the Navigation Model should be made (remember, with as much information as possible at the moment) and use them together should show the overall picture of the problem.

To present the methodology, the first type of artifact to have explained its preparation and use is the *Story Card*. Based on the example of a groupware Figure 2 shows the front of a Story Card requesting the management of pages in a wiki.

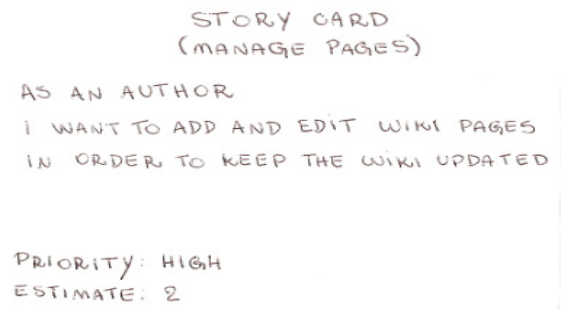

Figure 2: Example of a Story Card to a wiki.

As can be seen in the figure, the Story Card is as simple and objective as possible. The user can have a greater sense of being understood and can come to collaborate more with the identification of require-ments. But such information is of a very high level of abstraction, even if we consider what is written on the back of the Card, which will be shown later. The user said "what" expects that the application does, but not "how" expects that the application works.

This is where it is necessary to use the *Navigation Model*, to extract the user information about how the application will be used. One of the best ways to achieve this result is to draw the model with pen and paper, as show Figure 3, or on a whiteboard. This informality is a good way to encourage the User to interact.

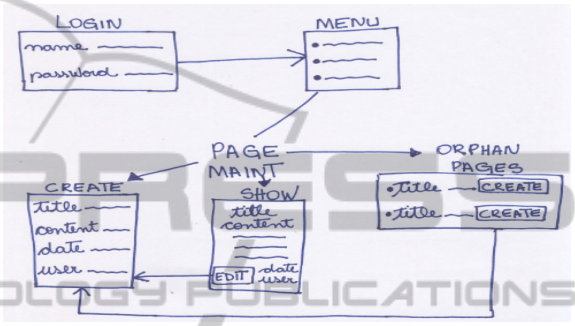

Figure 3: Navigation Model.

During these conversations, using both the Navigation Model and the Story Cards, the requirements are more detailed. This is the gain about using these two artifacts together. Rules that are not explicit in the Navigation Model, or even to make them clearer, may be written on the back of the corres-ponding Story Card, as can be seen in Figure 4. Other way to see these "requirements more detailed" in the back of the Story Card is like a form of confirmations. Confirmations are "questions" that may appear in a talk with the User, an iteration plan or in the implemen-tation time.

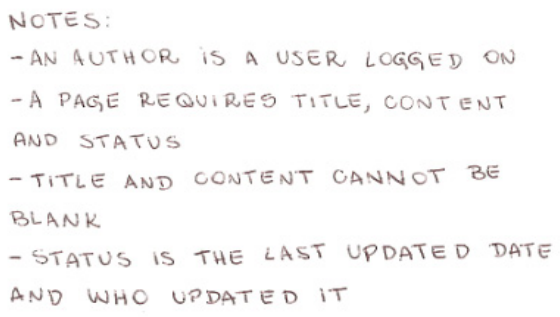

Figure 4: Back of the Story Card of Figure 2.

Importantly, the ideal to have as much information as possible in these conversations is to send the most experienced Web Application

Consultant with the skills necessary to interact with the user. If possible, the same consultant must have some knowledge in the user's business. Otherwise, send a Business Analyst with the consultant. These skills, besides assist in the extraction of requirements, are used to estimate the difficulty of accomplishing what is explicit in the union of the Story Cards and the Navigation Model. This estimate must be noted on the corresponding Story Card.

Another necessary note and extremely useful is "what" the User considers most urgent to be delivered in early iterations of the project, if possible. Therefore, each Story Card should have a priority and an estimate of difficulty (see Figure 2).

Nothing prevents other artifacts from be created in the search for more requirements. The more practical – but without losing content – the artifact is greater is the likelihood of having a user interaction. An example of this "practical artifact" is the Types Diagram (D'Souza, Wills, 1998) which is extremely simple, showing only the entities (with attributes) and their relation-ships. The user does not need to know what an entity is necessarily for example, just understand what the drawing represents and interact with it. The Types Diagram can also be used to specify information in a legacy database that will be used by Web application for example.

Technical issues such as infrastructure are not discussed in the Story Cards but they should be noted for later analysis by the development team. If these issues have a direct relationship with a Story is important to note it in the back of the corresponding Story Card.

At the end of the Communication, at least two types of artifacts (as shows in the Figure 1) should have been made: the Story Cards and the Navigation Model. Both serve to guide the next discipline.

3.2 Modeling

The development team, in this discipline, uses the Story Cards, the Navigation Model and any other artifacts that were created in *Communication* to create or redefine (in case of fixing) the application specifi-cations. They should be transcribed to an electronic format which means using any graphical tool or CASE tool.

The team should draw a *Class Diagram* or refine the Types Diagram previously created. This should be doing because this type of diagram helps the develop-ment team see what they will implement. The Figure 5 shows a more refined version of the Class Diagram for the wiki example.

This Class Diagram, initially much simple, is

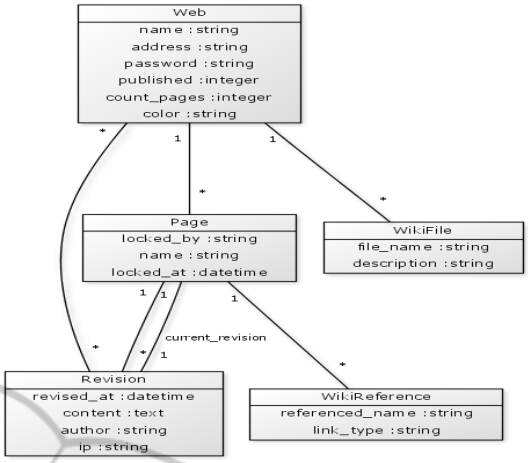

Figure 5: Refined version of the Class Diagram.

refined in each iteration. It is noteworthy that this diagram should represent how the requirements of the artifacts of the *Communication* will be treated. Like for example, the necessity of create an abstract class or an interface. It is the discretion of the team using the Class Diagram to create the database (if it is not a legacy database) or create a Database Model. Thus, the team can create a database as complete as possible to deal with the application that will be created in cycles. If the opposite occurs (create the database gradually, accor-ding to application needs at the moment), that will increase the chance of having to remodel the database and spend more resources fixing the problem.

The next step is write the *User Stories* based on the artifacts that were elaborate until now. These User Stories are written in a DSL (Domain-Specific Lan-guage), proposed by Dan North, which enables a simpler way to communicate with the user and obtain their understanding. The user can check the User Stories and approves them or not, for example. Or the team may assume that these Stories represent as faithfully as possible was expressed by the user in the Story Cards and the Model of Navigation. The Figure 6 shows a User Story transcribed from the Story Card in Figure 2, its back in Figure 4 and the Navigation Model in Figure 3.

A User Story transcribed in this language represents an executable description without losing the sense that was specified in the identification of requirements (in the previous discipline). This description provides an element that was implicit in the cards: the Scenario.

The Scenario is a using perspective in the User Story that contains one or more criteria for acceptance.

```
Feature: Manage pages in the wiki
    In order to keep the wiki updated
    As an author
    I want to add and edit pages

    Scenario: Create a page
        Given I am on the wiki creation page
         When I fill in "Title" with "Test - create page"
          And I fill in "Content" with "Checking the content..."
          And I press "Save"
         Then I should see the new page
          And I should see the revision date
          And I should see the author`s name

    Scenario: Update a page
        Given that I have created a page "First wiki page"
         When I press "Edit"
          And I fill in "Content" with "Updating the content..."
          And I press "Update"
         Then I should see the updated page
          And I should see the new revision date
          And I should see the authors name
```

Figure 6: A User Story based on previous artifacts.

A User Story will be finished when all the acceptance criteria for each Scenario are attended.

Despite the example shown above, it is important to realize that one way to improve the use of User Stories is create it only for business rules. The CRUD – Create, Read, Update and Delete – (Kilov, 1998), once well defined in a Class Diagram, can be generated automatically from various CASE tools. This helps in saving development time, not only in the coding of this part but also in the maintenance of this automatic code.

Analyzing the User Stories and the others artifacts, the team should look for what can be reused from previous applications or which may give rise to an artifact reusable in future applications of this domain. Also, it is valid identify where a pattern can be included to facilitate understanding and further maneuver of the application. The team can also raise questions about the information obtained in the *Communication* to better understand the issue if necessary. It is important that the user is willing to be asked to solve the doubts of the team.

Another team's function is to verify what non-functional requirements (which are the technical issues, which as mentioned in the end of *Communication*) are also necessary for a User Story to be considered complete. At last, the team must verify the behavior with the recycling of some component or framework with patterns adopted and between the User Stories and technical issues. In such cases, it is necessary to create "techniques stories" called *Tasks*. These Tasks have the same treatment of the User Stories. However, as the name suggests, they may not have as an origin a Story told by the user.

A person responsible for managing User Stories, to accept them or not, should create a *Product Backlog* (like the Product Owner on Scrum). This Backlog consists of the User Stories and the Tasks. To control the Product Backlog a good idea is to use a project management tool.

It is suggested that the User Stories and Tasks to be printed. It is valid put them together in a board visibility for the whole team like the style of KanBan (Anderson, 2010). It is also necessary to analyze which framework for acceptance tests will be used to perform the User Stories. Some of these frameworks are: Cucumber, JBehave, NBehave, easyb and Jasmine, among others. The choice should consider what the most appropriate framework for the programming language that will be adopted and how much work will be to integrate this framework in the process.

Another recommendation is to use patterns like the MVC – Model-View-Controller – (Loyd, Rimov, 2004), to better separate the layers between the data and interface (view). Another advantage of the MVC pattern is that the frameworks for execute the User Stories – which should be used in the next discipline – can generate error messages more comprehensible for each layer.

To determine what will be developed in each iteration must be taken into account the priorities established with the user (in *Communication*) and what is really possible to be done so that the iteration has the shortest duration possible. This will be the *Iteration Backlog* (similar to the *Sprint Backlog* on Scrum).

It is recommended (Pressman, Lowe, 2009) that one iteration in the development of a Web applications do not take more than a few weeks. This period of "few weeks", considering the guidelines of the Scrum Sprint, usually takes one to four weeks. All User Stories, and the relative Tasks, chosen to participate in the iteration must be completed within the cycle. This is a commitment from the development team.

Other decisions in the *Modeling* are related to hardware and software platforms adopted for the project. For example, it is defined the programming language will be Ruby. This language requires the Rails framework to be used in the Web development. So, in this case, it is necessary create an *Architecture Model* that shows how the MVC works with Ruby and Rails. Another decision is about which DBMS (DataBase Management System) will be used in the case of creating a database.

Thus, in the end of the *Modeling* discipline, as show in the process figure (Figure 1), the new artifacts are the User Stories transcribed, the Product Backlog, the Iteration Backlog, the Class Diagram and the Architecture Model. Like in *Communication*, nothing prevents other artifacts from being created in

this discipline to help the team comprehend the problem.

3.3 Construction

This discipline covers the coding and testing. By representing the implementation of Web application, this discipline can be considered as equivalent to the *Sprint* of the Scrum. However, this process goes into more detail on how the implementation should be made in search of an application closer to the ideal for the User. Practices such as *Daily Meeting* (Scrum), *Pair Programming*, *Continuous Integration* and *Standar-dized Code* (XP) are welcome, like any other practices that help the team to develop the Web application.

It encourages the use of tests before coding. Therefore, if possible, the tests (in addition to acceptance tests, which are the User Stories) must be created before any code to help plan what will be done. This concept is known as *Test Driven Development* (TDD). This is not mandatory in the process, but it is good practice and should be considered.

Based on these artifacts: Class Diagram, Architecture Model and the User Stories; the development team must create the necessary infrastructure to support the iteration. Probably the first iteration will need more activities, like to create the database with the information collected in *Communication* and analyzed in *Modeling*, for example. It is recommended to take this moment to generate all possible automatic codes and then write (and test) the code concerning about the business rules.

Making use of the Iteration Backlog, the development team must execute the User Stories under it and start coding based on the error messages that the chosen test framework for User Stories returns. Figure 7 shows the first error in the execution of the User Story of Figure 6.

At this point can be seen the need to have been adopted a framework for acceptance tests in *Modeling*. It is important to realize that by correcting the errors occurring in the implementation of User Stories, the development team can create features to the application based on a simulation of use by the User. And these are precisely the features most important to the application as it represents the business rules that the User must have in the application.

As previously mentioned, the use of TDD may be necessary to test methods that are part of one or more features requested by the User Stories. Therefore, these testing methods are of a lower-level of abstraction, being indicated not enter these kinds of details in the User Stories. Such tests prevail, for example, if the calculation of any monetary value is correct or even the proper use of the MVC layers to ensure future maintenance. Figure 8, based on a figure of The Book RSpec (Chelimsky, et al., 2010 Beta), shows the relationship between high-level of abstraction of a User Story and low-level of abstraction achieved using unit tests, for example.

The implementation process (preferably of business rules) starts with the tests being executed in a User Story. It is focused on one of the Scenarios, normally the first with errors (1). A step is written. This step refers to the acceptance criteria of the current Scenario (2). When the need arises for specialization, a unit test should be created. Any time a unit tests fails, for make it pass, it is necessary to go through TDD (3 to 5). As long as there are acceptance criteria unmet in the current scenario (6 and 7), the steps 2 to 7 should be re-run. When the Scenario is attended, the process restarts (1) for the next Scenario (as it is on the same User Story or on the next one). When there is no specialization (unit test), the cycle constitutes of only the User Story, with the activities 1, 2 (coding to pass this step) and 7. This process repeats itself iteratively, by creating tests for each acceptance criteria, of each Scenario, of each User Story. The code necessary is implemented, until the functional requirement for the User Story is fulfilled and all the tests (unit and acceptance) are passing. By the end of the iteration cycle, a fully functional prototype is ready.

```
(::) failed steps (::)

Can't find mapping from "the wiki creation page" to a path.
Now, go and add a mapping in /home/vinicius/Projects/wiki/features/support/paths.rb (RuntimeError)
./features/support/paths.rb:27:in `rescue in path_to'
./features/support/paths.rb:21:in `path_to'
./features/step_definitions/web_steps.rb:20:in `/^(?:|I )am on (.+)$/'
features/manage_pages.feature:7:in `Given I am on the wiki creation page'

Failing Scenarios:
cucumber features/manage_pages.feature:6 # Scenario: Create a page

2 scenarios (1 failed, 1 undefined)
14 steps (1 failed, 6 skipped, 7 undefined)
0m0.895s
```

Figure 7: First error when executing the User Story of Figure 6.

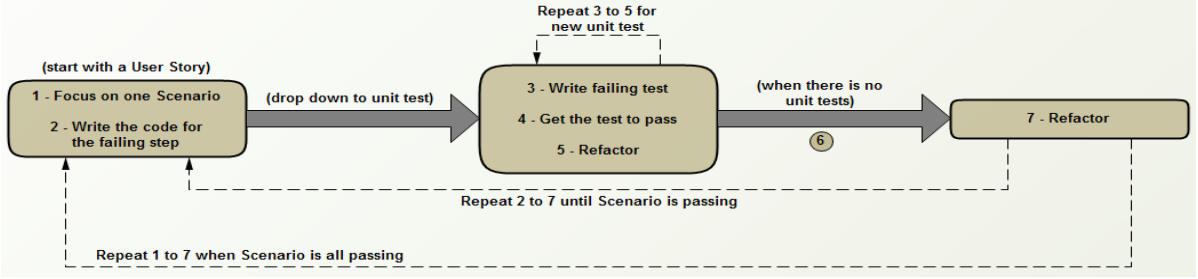

Figure 8: Relationship between high-level of abstraction (User Story) and low-level of abstraction (unit tests).

At the end of Construction, the prototype should be presented to the User analyze. Because it was created from simulations of use, the chance that the User accepts the product is great. Nothing prevents the User wants to putting this prototype into production. In this case, the prototype must be refined in relation to the design of interfaces because they are as simple as possible.

The User may or may not request changes. If he requests, the code leaner may be easier to be changed. Both changes as other features not seen before are treated in a new iteration, returning in the stage of *Communication*. If no change or no new functionality is required, the process returns to *Modeling*, where the team discusses what was done and seek improvements, besides selecting the next User Stories for the new iteration.

Nothing prevents the User requests to modify the priorities of the Story Cards (or even User stories) before iteration begins. If he requests during an iteration the team must negotiate the possibility of change it or not. User Stories, registered in the agile project management tool, may have changed his priorities, besides being marked as "delivered" or "not acceptable". Please note that the User shall, at all times, be aware of the Product Backlog and the current Iteration Backlog.

If the implementation involves changes in the *Modeling*, the artifacts must be updated to a new version, to make the documentation consistent. For example, the Navigation Model has navigation between pages different of what was previously specified. The updated artifacts (i.e., Class Diagram) can even be generated from code by reverse engineering.

4 RELATED WORKS

In addition to previously mentioned, other processes were analyzed. The process Agile Web Development with Web Framework (AWDWF) (Ran, et al., 2008), according to the authors, consists in use frameworks to guide the Project discipline of traditional Software Engineering. The proposal is to use the frameworks for the part of the development that is not linked to business logic, thus increasing the stability and efficiency, and ensure quality in Web application. This way, the team focus stays in the business logic. Also, cites that the communication with the Customer must be continuous.

In comparison, the procedure proposed in this article, in addition to using frameworks, show how to interact with the Customer and how to use the artifacts of this interaction to guide the coding of Web application. It also shows the need to write tests before the code and the gain that it generates.

Another case considered is an improvement in the life cycle of XP with added an activity of project management (Altarawneh, El Shiekh, 2008). This addition, it is justified because the project management of XP to be considered limited. The process is separated into six steps: (1) start with small Web projects, (2) adopt the modified XP process, (3) apply XPMM – eXtreme Programming Maturity Model – (Nawrocki, Walter, 2001), (4) education and training, (5) internal evaluation and lighter formal reviews and (6) external evaluation. All these steps should be implemented with the best practices in Web Engineering.

The proposed process is based not only on eXtreme Programming, but also in other methods such as Pressman's Web Engineering and Scrum. Another difference is that the process presented uses the concept of behavior driven development, to the tests be as close as possible to the User expects. Furthermore, the process is based on the concept of test driven development to generate the minimal code needed for the Web application be built without unnecessary functions.

In the Agile Software Engineering Environment (Aoyama, 1998) is shown an integrated software engineering environment developed to manage what he labeled as the Agile Software Process (Aoyama, 1998). The author presents a way of dealing (in a more high-level) with an agile development. It is an

excellent and comprehensive way to manage an agile project.

However, this proposal does not demonstrate the low-level part of the development (the coding) with more details, which is what the methodology proposed here aims to do. In the opinion of the authors to ensure that what was requested by User, in the specification of requirements, is well developed by the team has to have a direct relationship between requirements and code, that is, as the first directly implies the second.

UWE – Uml-based Web Engineering – (Knapp, Koch, Wirsing, Zhang, 2007) is a software engineering approach to the development of Web applications. It provides a UML profile (extension of UML), a meta model, a process of model-driven development and a support tool – Argo UWE – for the design of Web applications.

This proposal is a form to adapt UML in the Web context and although have a good support tool it doesn't have the support for the non-automatic code (business rules) that all application needs. This kind of support is what the RAMBUS try to have using the User Stories to guide the development and, with this, have a more close relationship among the disciplines – from requirements to code, passing through all the artifacts needed.

5 CONCLUSIONS

The agile methodology for Web development proposed here is a way to connect the areas of Requirements (Communication), Analysis and Design (Modeling), and Implementation (Construction) through artifacts that are highly related. To write and test the application code in *Construction* is necessary to run a User Story, which was created in *Modeling*. The way that the User Story has been created it will be used. The same thinking applies to the creation of the User Story. It was created from the requirements of *Communication*, through the Story Cards and Navigation Model. The methodology thus shows an efficient way to prove that the condition requested by the User was in fact implemented.

There are limits to this approach. The larger the system, the greater will be the difficulty to deal with Story Cards and create the Model Navigator, and later the User Stories, besides the fact that it consuming more time in conversations with the User. Thus, the methodology is suitable for Web applications to small and medium businesses. Another limit is the fact that non-functional quality attributes are hard to be placed as User Stories.

Future works includes the refinement and a more formal approach, with the study of a form to deal with the guarantee of quality in non-functional attributes and maintenance of the whole project, besides the construction of tools and other resources to support the process. A case study will also be conducted to prove the viability of the process and collect data such as effort, time and level of maintenance compared to other agile methods.

ACKNOWLEDGEMENTS

The authors would like to thanks the people from their laboratory for all the support and cooperation.

REFERENCES

Altarawneh, H., El Shiekh, A., 2008. A Theoretical Agile Process Framework for WebApplications Development in Small Software Firms. *6th SERA*.

Anderson, D., 2010. *KanBan*. Blue Hole Press. 1st Edition.

Aoyama, M., 1998. Agile Software Process and Its Experience. *20th ICSE*, IEEE Computer Soc. Press, pp 3--12.

Aoyama, M., 1998. Web-Based Agile Software Development. *IEEE Software*, Volume 15, Issue 6, pp 56--65.

Beck, K., Andres, C., 2004. *Extreme Programming Explained: Embrace Change*. Addison-Wesley. 2nd Edition.

Beck, K., 2003. *Test Driven Development by Example*. Addison Wesley. 1st Edition.

Carstensen, P. H., Schmidt, K., 1999. Computer supported cooperative work: new challenges to systems design. *Handbook of Human Factors*. 1st Edition.

Chelimsky, D., Astels, D., Dennis, Z., Hellesøy, A., Helmkamp, B., North, D., 2010. *The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber, and Friends*. Pragmatic Bookshelf. Beta Edition.

Cohn, M., 2004. *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional. 1st Edition.

D'Souza, D. F., Wills, A. C., 1998. *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach*. Addison-Wesley Professional. 1st Edition.

Deshpande, Y., Hansen, S., 2001. Web Engineering: Creating Discipline among Disciplines. *IEEE Multimedia*, Vol. 8, Number 1, pp 81--86.

Feathres, M., 2004. *Working Effectively with Legacy Code. Prentice Hall*. 2nd Edition.

Fowler, M., 1999. *Refactoring: Improving the design of existing code*. Addison-Wesley. 1st Edition.

Ginige, A., Murugesan, S., 2001. Web Engineering: An Introduction. *IEEE Multimedia*, Vol. 8, Number 1, pp 14--18.

Knapp, A.; Koch, N.; Wirsing, M.; Zhang, G., 2007. UWE

- Approach to Model-Driven Development of Web Applications. *i-com Journal*, v. 3, p. 3-12, Oldenbourg, Germany.
Kappel, G., Proll, B., Seiegfried, Retschitzegger, W., 2003. *An Introduction to Web Engineering*. John Wiley and Sons.
Kilov, H., 1998. *Business Specifications: The Key to Successful Software Engineering*. Prentice Hall. 1st Edition.
Loyd, D., Rimov, M., 2004. *Expresso Developer's Guide*. JCorporate Ltd.
Marca, D. A., McGowan, C. L., 1988. *SADT – Structured Analysis and Design Technique*. McGraw-Hill.
Martin, R., 2008. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR. 1st Edition.
Nawrocki, J. R., Walter, B., 2001. Toward Maturity Model for eXtreme Programming. *27th Euromicro Conference: A Net Odyssey*, pp 0233.
North, D., 2006. *Introducing Behaviour Driven Development*. Better Software. 1st Edition.
O'Reilly, T., 2006. Web 2.0 Compact Definition: Trying Again. O'Reilly Network.
Pressman, R., 2001. What a Tangled Web we Weave. *IEEE Software*, Vol. 18, Number 1, pp 18—21.
Pressman, R., Lowe, D., 2009. *Web Engineering: A Practitioner's Approach*. The McGraw-Hill Companies, Inc. 1st Edition.
Ran, H., Zhuo, W., Jun, H., Jiafeng, X., Jun, X., 2008. Agile Web Development with Web Framework. *4th WiCOM*.
Recuero, R. C., 2004. Redes sociais na Internet: Considerações iniciais. *XXVII INTERCOM*.
Schwaber, K., 2004. *Agile Project Management with Scrum*. Microsoft Press.